OIPE JC130
OCT 07 2004
PATENT & TRADEMARK OFFICE

AF
IFW

[40101/09301]

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s)	:	David N. WILNER et al.
Serial No.	:	09/480,309
Filing Date	:	January 10, 2000
For	:	PROTECTION DOMAINS FOR A COMPUTER OPERATING SYSTEM
Group Art Unit:	:	2127
Examiner	:	S. Ali

Mail Stop: Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Certificate of Mailing

I hereby certify that this correspondence is being deposited with U.S. Postal Services as first class mail in an envelope addressed to:

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

By: [illegible] Date: October 4, 2004
Michael J. Marcin (Reg. No. 48,198)

TRANSMITTAL

Transmitted herewith for filing in the above-identified patent application is an Appeal Brief under 37 C.F.R. § 1.192. No fees are believed to be required. However, if any fees are due, The Commissioner is hereby authorized to charge the Deposit Account of **Fay Kaplun & Marcin, LLP No. 50-1492.** A copy of this paper is enclosed for that purpose.

Respectfully submitted,

Dated: October 4, 2004

By: [illegible]
Michael J. Marcin (Reg. No. 48,198)

FAY KAPLUN & MARCIN, LLP
150 Broadway, Suite 702
New York, NY 10038

PATENT
Attorney Docket No.: 40101-09301

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Application of:	)	
	)	
David N. Wilner, et al.	)	
	)	
Serial No.: 09/480,309	)	Group Art Unit: 2127
	)	
Filed: January 10, 2000	)	Examiner: Syed J. Ali
	)	
For: PROTECTION DOMAINS FOR A COMPUTER OPERATING SYSTEM	)))	**Board of Patent Appeals and Interferences**

Commissioner for Patents
P.O. Box 1450
Arlington, VA 22313-1450

Sir:

APPEAL BRIEF UNDER 37 C.F.R. § 1.192

In support of the Notice of Appeal filed August 3, 2004, and pursuant to 37 C.F.R. § 1.192, appellants present in triplicate their appeal brief in the above-captioned application.

This is an appeal to the Board of Patent Appeals and Interferences from the Examiner's final rejection of claims 1-34 in the final Office Action dated March 10, 2004. The appealed claims are set forth in the attached Appendix A.

1. Real Party in Interest

This application is assigned to Wind River Systems, Inc., the real party in interest.

2. Related Appeals and Interferences

There are no other appeals or interferences which would directly affect, be directly affected, or have a bearing on the instant appeal.

3. Status of the Claims

Claims 1-34 have been rejected in the final Office Action.

Applicants submitted an Amendment After Final on August 3, 2004 which cancelled claims 4, 6, 7, 11, 12, 24 and 38-34 and amended claims 1, 5, 8, 13-15, 25 and 26. The amendments added limitations from the dependent claims into the independent claims. The Examiner has not issued an Advisory Action based on this Amendment. However, in a phone conversation with Applicants' attorney, the Examiner indicated he would enter the amendments. Thus, claims 1-3, 5, 8-10, 13-23 and 25-27 as presented in the August 3, 2004 Amendment After Final are involved in this appeal.

4. Status of Amendments

All amendments submitted by the appellants have been entered.

5. Summary of the Invention

The present invention comprises a protection domain system which allows an operating system to dynamically allocate system resources among processes and flexibly implements and enforces a protection hierarchy. (See Specification, page 12, lines 18-20). The operating system 112 may be object oriented wherein a protection domain 120 is actually a system object. (See Specification, page 14, lines 18-23). The protection domain 120 can be created, modified or destroyed by the operating system 112 or by a user task via an application programming interface provided by the operating system 112. (See Specification, page 14, line 22 - page 15, line 3). Included in the protection domain 120 may be a memory space 122, a protection view 123, zero or more code modules 126 containing executable code and/or data structures, a collection of protection domain attributes 127, a linking table 128, a symbol table 129, a list of entry points 130, zero or more tasks 124 and zero or more system objects 125 (e.g., semaphores, file descriptors, message queues, watchdogs, etc.). (See Specification, page 15, lines 5-20). The protection view 123 represents the protection domains to which tasks executing in the protection domain 120 may have access. (See Specification, page 16, lines 4-16). The code module 126 may contain executable code, data structures and/or other information to be used by a loader application during a loading process. (See Specification, page 17, lines 19-23). The executable code may include instructions that reference other executable code outside of the code module 126 using, for example, a symbol references. (See Specification, page 17, line 23 - page 18, line 5).

The symbol that references a location external to the protection domain 120 indicates that a link is needed between the protection domain 120 and another protection domain that owns the external location referenced by the symbol. (See Specification, page 18, lines 12-15). A linking table 128 is provided which includes an entry 132 for each symbol that requires an inter-protection domain link. (See Specification, page 18, lines 14-16). The entry 132 may include the symbol and a link stub 133, which comprises a number of executable instructions used to access the external location including, for example, a jump instruction to cause a jump to the address of the external location. (See Specification, page 18, lines 17-21).

Utilizing the above-described elements, the present invention allows the protection domain system to dynamically define a linkage between protection domains. In this manner, a task 705 is created in protection domain 702 to run an executable code of an application. (See Specification, page 39, lines 13-15). Task 705 has a protection view identical to that of the protection domain 702, which includes protection domains 703 and 704. (See Specification, page 39, lines 14-15). Task 705 may call a function in protection domains 703 and 704 by an unprotected link, because they are both within the protection view of task 705. (See Specification, page 39, lines 16-18). Use of the unprotected link means that the protection view of task 705 is altered to include protection domains 703 and 704. (See Specification, page 39, lines 20-23). However, without task 705, executable code in protection domain 703 could only access protection domain 704 by a protected link. (See Specification, page 39, line 23 - page 40, line 5). If the protected link is established, the protection view of the protection domain 703 is not altered, but a corresponding address for a symbol is inserted into a link stub of a linking table entry for the symbol. (See Specification, page 40, lines 1-5). Using the protected link, a memory address for the symbol will be outside of a memory accessible by the protection domain 703 of a calling code module in protection domain 703. (See Specification, page 40, lines 1-5).

6. Issues

I. Whether claims 1-3 and 5 are unpatentable under 35 U.S.C. § 103(a) as obvious over U.S. Patent No. 5,359,721 to Kempf et al. ("the Kempf reference") in view of U.S. Patent No. 6,546,546 to Van Doorn ("the Van Doorn reference").

II. Whether claims 8-10 and 13-14 are unpatentable under 35 U.S.C. § 103(a) as obvious over the Kempf reference in view of U.S. Patent No. 6,199,152 to Kelly et al. ("the Kelly reference") in further view of the Van Doorn reference.

III. Whether claims 15-23 and 25-27 are unpatentable under 35 U.S.C. § 103(a) as obvious over the Kempf reference in view of the Van Doorn reference.

7. Grouping of Claims

Claims 1-3, 5, 8-10, 13-23 and 25-27 may stand or fall together.

8. Argument

I. The Rejection of Claims 1-3 and 5 Under 35 U.S.C. § 103(a) as Being Obvious Over U.S. Patent No. 5,359,721 to Kempf et al. in view of U.S. Patent No. 6,546,546 to Van Doorn Should Be Reversed.

A. The Examiner's Rejection

Appellants amended claim 1, after the final rejection, to include all the recitations of cancelled claim 4. (See Amendment After Final, 8/3/04, page 9). The Examiner has since indicated that this amendment would be entered, because it did not require a new search. (Teleconference with Examiner, 9/23/04). Appellants would like to thank the Examiner for his consideration and entry of the after-final amendment. In view of the after-final amendment, the final rejection of claim 4 will be discussed.

In the final office action, the Examiner has rejected claims 1-3 and 5 under 35 U.S.C. 103(a) as being unpatentable over the Kempf reference in view of the Van Doorn reference. (See 3/10/04 Office Action, ¶ 5). The Kempf reference describes a method for a process executing in non-supervisor mode to perform cross address space dynamic linking. (See the Kempf reference, col. 4, lines 65-67). Address space is managed through an address space manager, which makes the address space available to a client process. (See the Kempf reference, col. 6, lines 54-56). The Van Doorn reference describes a Java Virtual Machine (JVM), which uses hardware protection domains to transparently and securely protect Java applets. (See the Van Doorn reference, col. 2, lines 43-44). In order to access methods not within a protection domain of a caller, a cross protection domain call must be performed, which must pass through a Java Nucleus to control access and use CPU resources. (See the Van Doorn reference, col. 8, lines 20-22).

B. The Cited References Do Not Disclose Requesting Attachment of the Second Domain to the First Domain When the Second Domain is Determined Not to be Within The Protection View of the First Domain and Attaching the Second Domain to the First Domain Using an Attachment Mechanism as Recited in Claim 1

The Examiner acknowledged in the Final rejection that the Kempf reference fails to disclose the steps of "requesting attachment of the second domain to the first domain when the second domain is determined not to be within the protection view of the first domain" and "attaching the second domain to the first domain using an attachment mechanism." (See 3/10/04 Office Action, ¶ 5). However, the Examiner further stated that the Van Doorn reference shows these claimed elements, thereby rendering the claimed subject matter obvious over the Kempf reference in view of the Van Doorn reference. (See 3/10/04 Office Action, ¶ 5). Appellants respectfully disagree with the Examiner's rejection of claim 1.

A protection domain, as taught by the present specification, is a container for system resources, executable code and data structures, as well as for executing tasks and system objects (e.g., semaphores, message queues, etc.). (See Specification, page 10, lines 19-21). Each resource and object in the system is owned by one protection domain. (See Specification, page 10, lines 22-23). This isolation prevents the task executing in the protection domain from interfering with resources or objects owned by other protection domains. (See Specification, page 11, lines 1-3). However, each protection domain has a protection view, which defines the system resources and objects to which it has access (i.e., within the protection domain itself and the other protection domains). (See Specification, page 11, lines 8-11). The system provides for an attachment mechanism, whereby the task executing in the protection domain may access the resources and the objects contained in the other protection domains. (See Specification, page 11, lines 5-8). The attaching mechanism is described as follows:

> When a first protection domain has obtained "unprotected attachment" to a second protection domain, the second protection domain is added to the protection view of the first protection domain. Executable code in the first protection domain may use "unprotected links" to selected functions in the second protection domain, allowing tasks executing in the first protection domain to use the resources and access the objects of the second protection domain with a minimum of execution overhead. (See Specification, page 11, lines 12-18).

The system also provides for a "protected attachment," which does not add the second protection domain to the protection view of the first protection domain:

> Using the protected link, a task running in the first domain may, for example, make a direct function call to a function existing in the second protection domain. Tasks in the first protection domain are prevented from accessing the second protection domain except through this protected link, thus preventing unauthorized accesses of functions and data in the second protection domain. (See Specification, page 12, lines 8-13).

In operation, a code module in the first protection domain may contain a reference to an executable code or data structure outside of the code module. (See Specification, page 18, lines 1-2). The reference may be made using a symbol, which represents a memory location of the executable code or data structure. (See Specification, page 18, lines 2-3). When the code module is executed and the symbol is found in the second protection domain which is in the protection view of the first protection domain, the unprotected link is established between the first and second protection domain. (See Specification, page 29, lines 4-24). A corresponding address for the symbol in the second protection domain is inserted into the first protection domain into a link stub, which provides a number of executable instructions (i.e., jump instructions) to access the address in the second protection domain. (See Specification, page 29, lines 20-22). However, if the symbol is found in the second protection domain which is not in the protection view of the first protection domain, the protected link may be established in the same manner as stated above regarding the link stub, but the protection view of the first protection domain is not altered. (See Specification, page 10, lines 19-21). Thus, the memory address for the symbol will be outside of the memory accessible by the code module in the first protection domain.

The Examiner has acknowledged that the Kempf reference does not disclose or suggest "requesting attachment of the second domain to the first domain when the second domain is determined not to be within the protection view of the first domain" and "attaching the second domain to the first domain using an attachment mechanism." (See 3/10/04 Office Action, ¶ 5). In the rejection, the Examiner further stated:

> a code module may need to access data that is outside of its particular protection domain. In such cases, security measures typically do not allow such access since it may cause a breach. This is the deficiency of Kempf. (See 3/10/04 Office Action, ¶ 5).

The deficiency of the Kempf reference was cured, according to the Examiner, by the teaching of the Van Doorn reference, because the Van Doorn reference purported to show that a code module can access data outside of its protection domain without restriction by security measures. (See

3/10/04 Office Action, ¶ 5).

Appellants respectfully submit that the Van Doorn reference fails to disclose or suggest the claimed steps of "requesting attachment of the second domain to the first domain when the second domain is determined not to be within the protection view of the first domain" and "attaching the second domain to the first domain using an attachment mechanism." The Van Doorn reference describes the JVM using hardware protection domains to protect Java applets. (See the Van Doorn reference, col. 2, lines 43-44). The "protection domains" are described as "a mapping of virtual to physical pages together with a set of domain specific events." (See the Van Doorn reference, col. 4, lines 64-65). Each protection domain has a view of its own subtree of the name space. (See the Van Doorn reference, col. 5, lines 23-25). The view includes a set of physical memory pages to virtual mappings. (See the Van Doorn reference, col. 8, lines 47-49). In order to access methods not within a caller's protection domain, a cross protection domain call must be performed which passes through a Java Nucleus to control access and use CPU resources. (See the Van Doorn reference, col. 8, lines 20-22).

The Van Doorn reference further teaches that the Java Nucleus controls access to different domains. The Java Nucleus is described as:

> analogous to an operating system kernel. It manages a number of protection domains and has full access to all memory mapped into these domains and their corresponding access permissions. *The protection domains themselves cannot manipulate the memory mappings or the access rights of their virtual memory pages. The Java Nucleus handles both data and instruction access (i.e., page) fault for these domains.* (See the Van Doorn reference, col. 8, lines 52-60) (emphasis supplied).

The exclusive control over the protection domains exercised by the Java Nucleus is described in the example provided in the Van Doorn reference:

> Consider the protection Domains A and B and a Method M which resides in Domain-B. An instruction fault occurs when A calls Method M, since M is not mapped into context A. The fault causes an event to be raised in the Java Nucleus. The event handler for this fault is passed two arguments: the fault address (i.e., method address) and the fault location (i.e., call instruction). Using the method address, the Java Nucleus determines the method information which contains the destination domain and the access control information. Paramecium's event interface is used to determine the caller domain. Based on this information, an access decision is made [by the Java Nucleus]. (See the Van Doorn reference, col. 9, lines 48-60).

The entire thrust of the Van Doorn reference is for the Java Nucleus to broker access between the various protection domains. (See the Van Doorn reference, col. 8, line 52 - col. 9, line 30; col. 9, line 40 - col. 10, line 60). Thus, the Java Nucleus controls access to the resources of other protection domains, and the view of a requesting protection domain is never altered.

Appellants' recitation in claim 1 that a second domain is attached to a first domain is in direct contrast with the disclosure of the Van Doorn reference. According to the present invention, a protection view of the first (requesting) protection domain is altered to include the symbol's (second) protection domain. (See Specification, page, 30, lines 12-14). By adding the second protection domain to the protection view of the first protection domain, the memory address for the symbol will be inside the memory accessible by the first protection domain, which includes a calling code module. (See Specification, pages 30, line 21 - page 31, line 2). The protection view of the present invention can be altered to provide access to the second protection domain, whereas, in the Van Doorn reference, the protection domains have no control over access to each other. In the Van Doorn reference, the Java Nucleus, operating in the extensible operating system, is the sole medium of access between domains, because all "[i]nvocations of the other methods in other domains pass through the Java Nucleus." (See the Van Doorn reference, col. 6, lines 24-25). Thus, the Van Doorn reference does not disclose or suggest "requesting attachment of the second domain to the first domain when the second domain is determined not to be within the protection view of the first domain" and "attaching the second domain to the first domain using an attachment mechanism," as recited in claim 1.

The appellants also respectfully submit that there is no motivation to combine the Kempf reference with the Van Doorn reference. The Kempf reference includes an operating system 32, which "provides well known operating system services." (See the Kempf reference, col. 5, lines 41-42). However, the Kempf reference does not go any further in disclosing representative examples of the operating systems usable with the invention described therein. Also, a set of hardware elements utilized in the Kempf reference are "those typically found in most general purpose computer systems" and "are intended to be representative of...[a] broad category of data processing systems." (See the Kempf reference, col. 5, lines 41-42). Disclosed examples of the data processing systems are "computer systems manufactured by Sun Microsystems, Inc."

In stark contrast, the Van Doorn reference discloses a Java Virtual Machine (JVM), which consists of a Java Nucleus process that loads Java classes into separate hardware protection domains and mediates access between them. (See the Van Doorn reference, col. 2, lines 42-51). The JVM is applied to the integration of a programming language system and an operating system, where the operating system "has the property of including the ability to have low-level control over hardware protection domains, fault event redirection, and efficient IPC [interprocess communication]." (See the Van Doorn reference, col. 3, lines 48-65). The Java Nucleus, which is included in the JVM, is described as being "[c]entral to the protection domain scheme," because "[i]t manages a number of protection domains and has full access to all memory mapped into these domains and their corresponding address permissions." (See the Van Doorn reference, col. 8, lines 53-56). However, the operability of and benefit derived from the Java Nucleus, and therefore the teachings of the Van Doorn reference, "depends on user accessible low-level operating system functionality *that is currently **only** provided by extensible operating systems (e.g., Paramecium, L4/LavaOS, ExOS, and SPIN).*" (See the Van Doorn reference, col. 17, lines 56-58) (emphasis supplied).

In view of the limited operating systems that are usable with the Java Nucleus described in the Van Doorn reference, one of ordinary skill in the art would not have been motivated to combine the features of the Van Doorn reference with the teachings of the Kempf reference, which include an operating system that "provides well known operating system services." As there is no motivation to combine the references, Appellants respectfully request that the Board reverse the Examiner's rejection of claim 1.

Therefore, at least for these reasons, it is respectfully submitted that claim 1 is allowable. Appellants respectfully submit that the Board overturn the Examiner's rejection under 35 U.S.C. 103(a) of independent claim 1 and all the claims depending therefrom (claims 2, 3 and 5).

II. The Rejection of Claims 8-10 and 13-14 Under 35 U.S.C. § 103(a) as Being Obvious Over the Kempf Reference in View of U.S. Patent No. 6,199,152 to Kelly et al. ("the Kelly reference") in Further View of the Van Doorn Reference Should Be Reversed.

A. The Examiner's Rejection

Appellants amended claim 8, after the final rejection, to include all the recitations of cancelled claims 11 and 12. (See Amendment After Final, 8/3/04, page 10). The Examiner has since indicated that this amendment would be entered, because it did not require a new search. (Teleconference with Examiner, 9/23/04). Appellants would like to thank the Examiner for his consideration and entry of the after-final amendment. In view of the after-final amendment, the final rejection of claim 12 will be discussed.

In the final office action, the Examiner has rejected claims 8-10 and 13-14 under 35 U.S.C. 103(a) as being unpatentable over the Kempf reference in view of the Kelly reference in further view of the Van Doorn reference. (See 3/10/04 Office Action, ¶ 6). The Kempf reference and the Van Doorn reference have been discussed above. The Kelly reference describes a microprocessor having a "morph host" hardware portion and an emulating software portion, such that various microprocessor architectures can be emulated. (See the Kelly reference, col. 11, lines 6-15). One of the features of the microprocessor is an exception handling process that emulates exception handling on the microprocessor. (See the Kelly reference, col. 13, lines 8-61). The Kelly reference states that exceptions generated by the target software are to be handled by emulating the target processor hardware facilities using the code morphing software. (See the Kelly reference, col. 13, lines 42-48).

B. The Cited References Do Not Disclose, for at Least the Same Reasons Described Above, the Limitations, as Recited in Claim 8

Claim 8 recites "*altering the task protection view to include a protection view of the second domain.*" This limitation is analogous to the limitation recited in claim 1 and described above in Section 8.I.B. The Kelly reference neither teaches nor suggests such a limitation. Accordingly, for the same reasons described above in regards to claim 1, the Board should overturn the Examiner's rejection of claim 8 under 35 U.S.C. § 103(a) and all the rejected claims dependent therefrom (claims 9-10 and 13-14).

III. The Rejection of Claims 15-23 and 25-27 under 35 U.S.C. § 103(a) as Being Obvious Over the Kempf Reference in View of the Van Doorn Reference Should Be Reversed.

A. The Examiner's Rejection

Appellants amended claim 15, after the final rejection, to include all the recitations of cancelled claim 24. (See Amendment After Final, 8/3/04, page 10). The Examiner has since indicated that this amendment would be entered, because it did not require a new search. (Teleconference with Examiner, 9/23/04). Appellants would like to thank the Examiner for his consideration and entry of the after-final amendment. In view of the after-final amendment, the final rejection of claim 24 will be discussed.

In the final office action, the Examiner has rejected claims 15-23 and 25-27 under 35 U.S.C. 103(a) as being unpatentable over the Kempf reference in view the Van Doorn reference. (See 3/10/04 Office Action, ¶ 5). The Kempf reference and the Van Doorn reference have been discussed above.

B. The Cited References Do Not Disclose, for at Least the Same Reasons Described Above, the Limitations, as Recited in Claim 15

Claim 15 recites "*a number of protection domains, at least one of the number of protection domains owning a portion of the system space, wherein each of the number of protection domains includes a protection view defining a set of the number of protection domains to which unprotected access may be made.*" This limitation is analogous to the limitation recited in claim 1 and described above in Section 8.I.B. Specifically, the Van Doorn reference teaches that the protection domains include a view, but the Van Doorn reference lacks any teaching that the view includes other protection domains; the view is limited to its own domain. The Java Nucleus may have a view of the entire system, but the Java Nucleus is not one of the protection domains. Accordingly, for the same reasons described above in regards to claim 1, the Board should overturn the Examiner's rejection of claim 15 under 35 U.S.C. § 103(a) and all the rejected claims dependent therefrom (claims 16-23 and 25-27).

9. Conclusions

For the reasons set forth above, Appellants respectfully request that the Board reverse the final rejections of the claims by the Examiner under 35 U.S.C. § 103(a) and indicate that claims 1-3, 5, 8-10, 13-23 and 25-27 are allowable.

Respectfully submitted,

FAY, KAPLUN & MARCIN, L.L.P.

Date: 10/4/04

By: [illegible]
Michael J. Marcin
Reg. No. 48,198

APPENDIX A – APPEALED CLAIMS

1. A method, comprising the steps of:

loading a code module into a memory space of a first domain, the code module including an instruction having a symbol reference;

determining if the symbol reference is to an external location outside of the memory space;

generating a link stub for the symbol reference when the symbol reference is to an external location to access the external location;

redirecting the instruction to the link stub; and

determining if the external location is within a second domain that is within a protection view of the first domain;

requesting attachment of the second domain to the first domain when the second domain is determined not to be within the protection view of the first domain; and

attaching the second domain to the first domain using an attachment mechanism.

2. The method of claim 1, wherein the link stub is part of a linking table entry corresponding to the symbol reference.

3. The method of claim 1, wherein the link stub is a jump instruction to the external location.

5. The method of claim 1, further comprising the steps of:

determining whether the attachment request is permitted based on authorization information provided by the first domain;

wherein attachment to the second domain is not permitted when the attachment request is not permitted.

8. A method, comprising:

creating a task in a first domain, the task executing a number of instructions;

executing a first jump instruction in the number of instructions that refers to a link stub corresponding to an external location in a second domain;

executing the link stub;

comparing the external location to a task protection view;

generating a processing exception when the external location is outside the task protection view; and

executing an exception handling routine in response to the generation of the processing exception, the exception handling routine including,

saving a pre-exception setting of the task protection view,

altering the task protection view to include a protection view of the second domain, and

jumping to the external location.

9. The method of claim 8, wherein the link stub is part of linking table entry corresponding to the external location.

10. The method of claim 8, wherein the link stub includes a second jump instruction to the external location.

13. The method of claim 8, wherein the task protection view is saved on a task protection switch stack.

14. The method of claim 8, further comprising steps of:

retrieving the pre-exception setting of the task protection view;

restoring the task protection view using the pre-exception setting of the task protection view;

returning to an a subsequent instruction to the first jump instruction in the number of instructions.

15. A computer system, comprising

a system space having a number of memory locations;

a number of protection domains, at least one of the number of protection domains owning a portion of the system space, wherein each of the number of protection domains includes a protection view defining a set of the number of protection domains to which unprotected access may be made.

16. A computer system of claim 15, wherein the at least one of the number of protection domains includes at least one of

a code module,

a link stub, and

an entry point.

17. The system of claim 16, wherein the link stub is part of a linking table entry in a linking table.

18. The system of claim 16, wherein the entry point is represented in a symbol table.

19. The system of claim 16, wherein at least one of the number of protection domains is a system protection domain that includes:

at least one code module including executable code for operating system services; and

at least one system object owned by the system protection domain.

20. The system of claim 19, wherein the system protection domain includes a protection

domain list that includes entries for each of the number of protection domains.

21. The system of claim 19, wherein at least one of the number of protection domains is a first protection domain that includes:

at least one code module including executable code for a first set of functions; and

a number of link stubs;

wherein at least one of the link stubs corresponds to a symbol referenced in the executable code for the first set of functions, and such at least one link stub includes executable code to direct execution to the executable code for operating system services.

22. The system of claim 21, wherein the number of protection domains includes a second protection domain that includes:

at least one code module including executable code for a second set of functions; and

a number of entry points;

wherein each of the entry points corresponds to a symbol in the executable code for the second set of functions.

23. The system of claim 22, wherein one of the link stubs of the first protection domain corresponds to one of the number of entry points in the second protection domain, and such link stub includes executable code to direct execution to the one of the number of entry points in the second protection domain.

25. The system of claim 15, wherein the protection view sets a memory range of allowable memory accesses, and wherein a memory fault is generated when memory access is attempted outside of the memory range.

26. The system of claim 15, wherein the memory range is contiguous.

27. The system of claim 25, further comprising an exception handling routine that is executed on the occurrence of the memory fault, the exception handling routine including a protection switch mechanism.